EL977166301US

Docket No. AUS920030541US1

METHOD AND APPARATUS FOR COUNTING INTERRUPTS BY TYPE

CROSS REFERENCE TO RELATED APPLICATIONS

The present invention is related to the following applications entitled "Method and Apparatus for Counting Instruction Execution and Data Accesses", serial no. __________, attorney docket no. AUS920030477US1, filed on September 30, 2003; "Method and Apparatus for Selectively Counting Instructions and Data Accesses", serial no. __________, attorney docket no. AUS920030478US1, filed on September 30, 2003; "Method and Apparatus for Generating Interrupts Upon Execution of Marked Instructions and Upon Access to Marked Memory Locations", serial no. __________, attorney docket no. AUS920030479US1, filed on September 30, 2003; "Method and Apparatus for Counting Data Accesses and Instruction Executions that Exceed a Threshold", serial no. __________, attorney docket no. AUS920030480US1, filed on September 30, 2003; "Method and Apparatus for Counting Execution of Specific Instructions and Accesses to Specific Data Locations", serial no. __________, attorney docket no. AUS920030481US1, filed on September 30, 2003; "Method and Apparatus for Debug Support for Individual Instructions and Memory Locations", serial no. __________, attorney docket no. AUS920030482US1, filed on September 30, 2003; "Method and Apparatus to Autonomically Select Instructions for Selective Counting", serial no. __________, attorney docket no. AUS920030483US1, filed on September 30, 2003;

"Method and Apparatus to Autonomically Count Instruction Execution for Applications", serial no. __________, attorney docket no. AUS920030484US1, filed on September 30, 2003; "Method and Apparatus to Autonomically Take an Exception on Specified Instructions", serial no. __________, attorney docket no. AUS920030485US1, filed on September 30, 2003; "Method and Apparatus to Autonomically Profile Applications", serial no. __________, attorney docket no. AUS920030486US1, filed on September 30, 2003; "Method and Apparatus for Counting Instruction and Memory Location Ranges", serial no. __________, attorney docket no. AUS920030487US1, filed on September 30, 2003; "Method and Apparatus for Qualifying Collection of Performance Monitoring Events by Types of Interrupt When Interrupt Occurs", serial no. __________, attorney docket no. AUS920030540US1, filed on __________; and "Method and Apparatus for Providing Pre and Post Handlers for Recording Events", serial no. __________, attorney docket no. AUS920030543US1, filed on __________. All of the above related applications are assigned to the same assignee, and incorporated herein by reference.

BACKGROUND OF THE INVENTION

1. Technical Field:

The present invention relates generally to an improved data processing system and, in particular, to a method and system for monitoring the performance of the processor in a data processing system when an interrupt occurs. Still more particularly, the present invention relates to a method, apparatus, and computer instructions for counting interrupts by type.

2. Description of Related Art:

A typical data processing system utilizes processors to execute a set of instructions in order to perform a certain task, such as reading a specific character from the main memory. However, as the number of tasks required to be executed by the processor increases, the efficiency of the processor's access patterns to memory and the characteristics of such access become important factors for engineers seeking to optimize the system operation.

There are currently mechanisms in the prior art that can count occurrences of software-selectable events, such as cache misses, instructions executed, I/O data transfer requests, and the time a given process may take to execute within a data processing system. One such mechanism is a performance monitor. A performance monitor assists in performing an analysis of a system by monitoring selected characteristics and determining the state of the system at a particular time. The analysis may provide information of how the processor is used when instructions are executed and the processor's interaction with the main memory when data is stored. In addition, the analysis may provide detail regarding the amount of time that has passed between events occurring in the system. Thus, the performance monitor may be used to assist in analyzing system performance.

However, the performance monitor described above does not provide the ability to count a particular type of interrupt using hardware. An interrupt occurs when a device, such as, for example, a mouse or keyboard, raises an interrupt signal to notify the processor that an event has occurred. When the processor accepts an interrupt request, the processor completes its current instruction and passes control to an interrupt handler. The interrupt handler executes an interrupt service routine that is associated with the interrupt. An interrupt may also be caused by a specific machine language operation code, for example, Motorola 68000's TRAP, a product from Motorola, Inc. In this case, an unexpected software condition such as, for example, divide by zero causes the processor to store the current state, store identifying information about the particular interrupt, and pass control to an interrupt handler that handles this unexpected software condition.

An interrupt descriptor table (IDT) is a system table that associates each interrupt with corresponding interrupt handler containing corresponding interrupt service routines. The performance monitor described above also does not provide hardware support for counting interrupts by type. It would be advantageous to have an improved method, apparatus, and computer instructions for counting interrupts by type and storing the count in the IDT or outside of the IDT in an interrupt count table (ICT). In addition, it would be advantageous to have an improved method to support counting of interrupts by type using hardware instead of a software selectable performance monitor.

SUMMARY OF THE INVENTION

The present invention provides a method, apparatus, and computer instructions for counting interrupts by type. The mechanism of the present invention includes an interrupt unit that employs a hardware counter to count the interrupts and identify the interrupts by type. The interrupt count may be stored within a location in an interrupt descriptor table (IDT) or within an interrupt count table (ICT) outside of the IDT. If the count is stored in the IDT, the storage location is associated with the interrupt type, so that the interrupt unit of the present invention and associated supporting software have knowledge of the location within the IDT where the count is stored. Alternatively, if the count is stored outside of the IDT, such as in an interrupt count table (ICT), the interrupt unit uses the address of the ICT, stored in a register, and the interrupt type to determine the location to increment the count.

The present invention also provides logic necessary to detect if a count overflow occurs. If an overflow of the count occurs, the mechanism of the present invention allows the supporting software to handle the overflow. Once the count is recorded for a particular interrupt type, the supporting software reads and resets the count value so that the user may collect the result at a later time for performance analysis.

BRIEF DESCRIPTION OF THE DRAWINGS

The novel features believed characteristic of the invention are set forth in the appended claims. The invention itself, however, as well as a preferred mode of use, further objectives and advantages thereof, will best be understood by reference to the following detailed description of an illustrative embodiment when read in conjunction with the accompanying drawings, wherein:

Figure 1 is an exemplary block diagram of a data processing system in which the present invention may be implemented;

Figure 2 is an exemplary block diagram of a processor system for processing information according to a preferred embodiment of the present invention;

Figure 3 is an exemplary diagram illustrating components for counting interrupts by type in accordance with a preferred embodiment of the present invention;

Figure 4 is an exemplary diagram illustrating components for counting interrupts by type with the count stored outside of the IDT in accordance with a preferred embodiment of the present invention;

Figure 5 is an exemplary diagram illustrating components for counting interrupt by type with the count stored within the IDT in accordance with a preferred embodiment of the present invention; and

Figure 6 is a flowchart outlining an exemplary process of counting interrupts by type in accordance with a preferred embodiment of the present invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

The present invention provides a method, apparatus, and computer instructions for counting interrupts by type. The mechanism of the present invention for counting interrupts by type is preferably implemented using an interrupt unit. The interrupt count is incremented when a particular type of interrupt occurs. The interrupt count may be stored in an interrupt descriptor table (IDT) or an interrupt count table (ICT) outside of the IDT.

With reference now to **Figure 1**, an exemplary block diagram of a data processing system is shown in which the present invention may be implemented. Client **100** is an example of a computer, in which code or instructions implementing the processes of the present invention may be located. Client **100** employs a peripheral component interconnect (PCI) local bus architecture. Although the depicted example employs a PCI bus, other bus architectures such as Accelerated Graphics Port (AGP) and Industry Standard Architecture (ISA) may be used. Processor **102** and main memory **104** are connected to PCI local bus **106** through PCI bridge **108**. PCI bridge **108** also may include an integrated memory controller and cache memory for processor **102**. Additional connections to PCI local bus **106** may be made through direct component interconnection or through add-in boards. In the depicted example, local area network (LAN) adapter **110**, small computer system interface SCSI host bus adapter **112**, and expansion bus interface **114** are connected to PCI local bus

106 by direct component connection. In contrast, audio adapter **116**, graphics adapter **118**, and audio/video adapter **119** are connected to PCI local bus **106** by add-in boards inserted into expansion slots. Expansion bus interface **114** provides a connection for a keyboard and mouse adapter **120**, modem **122**, and additional memory **124**. SCSI host bus adapter **112** provides a connection for hard disk drive **126**, tape drive **128**, and CD-ROM drive **130**. Typical PCI local bus implementations will support three or four PCI expansion slots or add-in connectors.

An operating system runs on processor **102** and is used to coordinate and provide control of various components within data processing system **100** in **Figure 1**. The operating system may be a commercially available operating system such as Windows XP, which is available from Microsoft Corporation. An object oriented programming system such as Java may run in conjunction with the operating system and provides calls to the operating system from Java programs or applications executing on client **100**. "Java" is a trademark of Sun Microsystems, Inc. Instructions for the operating system, the object-oriented programming system, and applications or programs are located on storage devices, such as hard disk drive **126**, and may be loaded into main memory **104** for execution by processor **102**.

Those of ordinary skill in the art will appreciate that the hardware in **Figure 1** may vary depending on the implementation. Other internal hardware or peripheral devices, such as flash read-only memory (ROM), equivalent nonvolatile memory, or optical disk drives and the like, may be used in addition to or in place of the hardware depicted in **Figure 1**. Also, the processes of the present invention may be applied to a multiprocessor data processing system.

For example, client **100**, if optionally configured as a network computer, may not include SCSI host bus adapter **112**, hard disk drive **126**, tape drive **128**, and CD-ROM **130**. In that case, the computer, to be properly called a client computer, includes some type of network communication interface, such as LAN adapter **110**, modem **122**, or the like. As another example, client **100** may be a stand-alone system configured to be bootable without relying on some type of network communication interface, whether or not client **100** comprises some type of network communication interface. As a further example, client **100** may be a personal digital assistant (PDA), which is configured with ROM and/or flash ROM to provide non-volatile memory for storing operating system files and/or user-generated data. The depicted example in **Figure 1** and above-described examples are not meant to imply architectural limitations.

The processes of the present invention are performed by processor **102** using computer implemented instructions, which may be located in a memory such as, for example, main memory **104**, memory **124**, or in one or more peripheral devices **126-130**.

Turning next to **Figure 2**, an exemplary block diagram of a processor system for processing information is depicted in accordance with a preferred embodiment of the present invention. Processor **210** may be implemented as processor **102** in **Figure 1**.

In a preferred embodiment, processor **210** is a single integrated circuit superscalar microprocessor. Accordingly, as discussed further herein below, processor **210** includes various units, registers, buffers, memories, and other sections, all of which are formed by integrated circuitry. Also, in the preferred embodiment, processor **210** operates according to reduced instruction set computer ("RISC") techniques. As shown in **Figure 2**, system bus **211** is connected to a bus interface unit ("BIU") **212** of processor **210**. BIU **212** controls the transfer of information between processor **210** and system bus **211**.

BIU **212** is connected to an instruction cache **214** and to data cache **216** of processor **210**. Instruction cache **214** outputs instructions to sequencer unit **218**. In response to such instructions from instruction cache **214**, sequencer unit **218** selectively outputs instructions to other execution circuitry of processor **210**.

In addition to sequencer unit **218**, in the preferred embodiment, the execution circuitry of processor **210** includes multiple execution units, namely a branch unit **220**, a fixed-point unit A ("FXUA") **222**, a fixed-point unit B ("FXUB") **224**, a complex fixed-point unit ("CFXU") **226**, a load/store unit ("LSU") **228**, and a floating-point unit ("FPU") **230**. FXUA **222**, FXUB **224**, CFXU **226**, and LSU **228** input their source operand information from general-purpose architectural registers ("GPRs") **232** and fixed-point rename buffers **234**. Moreover, FXUA **222** and FXUB **224** input a "carry bit" from a carry bit ("CA") register **239.** FXUA **222,** FXUB **224,** CFXU **226,** and LSU **228** output results (destination operand information) of their operations for storage at selected entries in fixed-point rename buffers **234.** Also, CFXU **226** inputs and outputs source operand information and destination operand information to and from special-purpose register processing unit ("SPR unit") **237.**

FPU **230** inputs its source operand information from floating-point architectural registers ("FPRs") **236** and floating-point rename buffers **238.** FPU **230** outputs results (destination operand information) of its operation for storage at selected entries in floating-point rename buffers **238.**

In response to a Load instruction, LSU **228** inputs information from data cache **216** and copies such information to selected ones of rename buffers **234** and **238.** If such information is not stored in data cache **216,** then data cache **216** inputs (through BIU **212** and system bus **211**) such information from a system memory **239** connected to system bus **211.** Moreover, data cache **216** is able to output (through BIU **212** and system bus **211**) information from data cache **216** to system memory **239** connected to system bus **211.** In response to a Store instruction, LSU **228** inputs information from a selected one of GPRs **232** and FPRs **236** and copies such information to data cache **216.**

Sequencer unit **218** inputs and outputs information to and from GPRs **232** and FPRs **236.** From sequencer unit **218,** branch unit **220** inputs instructions and signals indicating a present state of processor **210.** In response to such instructions and signals, branch unit **220** outputs (to sequencer unit **218**) signals indicating suitable memory addresses storing a sequence of instructions for execution by processor **210.** In response to such signals from branch unit **220,** sequencer unit **218** inputs the indicated sequence of instructions from instruction cache **214.** If one or more of the sequence of instructions is not stored in instruction cache **214,** then instruction cache **214** inputs (through BIU **212** and system bus **211**) such instructions from system memory **239** connected to system bus **211.**

In response to the instructions input from instruction cache **214,** sequencer unit **218** selectively dispatches the instructions to selected ones of execution units **220, 222, 224, 226, 228,** and **230.** Each execution unit executes one or more instructions of a particular class of instructions. For example, FXUA **222** and FXUB **224** execute a first class of fixed-point mathematical operations on source operands, such as addition, subtraction, ANDing, ORing and XORing. CFXU **226** executes a second class of fixed-point operations on source operands, such as fixed-point multiplication and division. FPU **230** executes floating-point operations on source operands, such as floating-point multiplication and division.

As information is stored at a selected one of rename buffers **234,** such information is associated with a storage location (e.g. one of GPRs **232** or carry bit (CA) register **242**) as specified by the instruction for which the selected rename buffer is allocated. Information stored at a selected one of rename buffers **234** is copied to its associated one of GPRs **232** (or CA register **242**) in response to signals from sequencer unit **218**. Sequencer unit **218** directs such copying of information stored at a selected one of rename buffers **234** in response to "completing" the instruction that generated the information. Such copying is called "writeback."

As information is stored at a selected one of rename buffers **238**, such information is associated with one of FPRs **236**. Information stored at a selected one of rename buffers **238** is copied to its associated one of FPRs **236** in response to signals from sequencer unit **218**. Sequencer unit **218** directs such copying of information stored at a selected one of rename buffers **238** in response to "completing" the instruction that generated the information.

Processor **210** achieves high performance by processing multiple instructions simultaneously at various ones of execution units **220, 222, 224, 226, 228,** and **230**. Accordingly, each instruction is processed as a sequence of stages, each being executable in parallel with stages of other instructions. Such a technique is called "pipelining." In a significant aspect of the illustrative embodiment, an instruction is normally processed as six stages, namely fetch, decode, dispatch, execute, completion, and writeback.

In the fetch stage, sequencer unit **218** selectively inputs (from instruction cache **214**) one or more instructions from one or more memory addresses storing the sequence of instructions discussed further hereinabove in connection with branch unit **220**, and sequencer unit **218**.

In the decode stage, sequencer unit **218** decodes up to four fetched instructions.

In the dispatch stage, sequencer unit **218** selectively dispatches up to four decoded instructions to selected (in response to the decoding in the decode stage) ones of execution units **220, 222, 224, 226, 228,** and **230** after reserving rename buffer entries for the dispatched instructions' results (destination operand information). In the dispatch stage, operand information is supplied to the selected execution units for dispatched instructions. Processor **210** dispatches instructions in order of their programmed sequence.

In the execute stage, execution units execute their dispatched instructions and output results (destination operand information) of their operations for storage at selected entries in rename buffers **234** and rename buffers **238** as discussed further hereinabove. In this manner, processor **210** is able to execute instructions out-of-order relative to their programmed sequence.

In the completion stage, sequencer unit **218** indicates an instruction is "complete." Processor **210** "completes" instructions in order of their programmed sequence.

In the writeback stage, sequencer **218** directs the copying of information from rename buffers **234** and **238** to GPRs **232** and FPRs **236**, respectively. Sequencer unit **218** directs such copying of information stored at a selected rename buffer. Likewise, in the writeback stage of a particular instruction, processor **210** updates its architectural states in response to the particular instruction. Processor **210** processes the respective "writeback" stages of instructions in order of their programmed sequence. Processor **210** advantageously merges an instruction's completion stage and writeback stage in specified situations.

In the illustrative embodiment, each instruction requires one machine cycle to complete each of the stages of instruction processing. Nevertheless, some instructions (e.g., complex fixed-point instructions executed by CFXU **226**) may require more than one cycle. Accordingly, a variable delay may occur between a particular instruction's execution and completion stages in response to the variation in time required for completion of preceding instructions.

Completion buffer **248** is provided within sequencer **218** to track the completion of the multiple instructions which are being executed within the execution units. Upon an indication that an instruction or a group of instructions have been completed successfully, in an application specified sequential order, completion buffer **248** may be utilized to initiate the transfer of the results of those completed instructions to the associated general-purpose registers.

In addition, processor **210** also includes interrupt unit **250**, which is connected to instruction cache **214**. Additionally, although not shown in **Figure 2**, interrupt unit **250** is connected to other functional units within processor **210**. Interrupt unit **250** may receive signals from other functional units and initiate an action, such as starting an error handling or trap process. In these examples, interrupt unit **250** is employed to generate interrupts and exceptions that may occur during execution of a program. Interrupt unit **250** may also be employed to count the occurrence of interrupts by type and increment the count values accordingly.

As mentioned above, the present invention provides a method, apparatus, and computer instructions for counting interrupts by type. In other words, the present invention provides a mechanism for counting the number of times a particular interrupt type occurs. An interrupt type may be, for example, a specific interrupt vector associated with an interrupt, such as Virtual Hash Page Table (VHPT) instruction fault or a translation lookaside buffer (TLB) data fault. The present invention allows for counting the number of these example interrupt types occurring within a defined period, for example, a number of clock cycles. In addition, the interrupt unit of the present invention, such as interrupt unit **250** in **Figure 2**, uses hardware counters to increment the count rather than employing supporting software to perform the count as in the prior art.

Turning to **Figure 3**, an exemplary diagram illustrating components for counting interrupts by type is depicted in accordance with a preferred embodiment of the present invention. In this example implementation, central processing unit (CPU) **302** may be implemented as processor **210** in **Figure 2**. When an interrupt occurs, CPU

302 sends a signal to interrupt unit **304** to request interrupt unit **304** generate an interrupt signal for a particular interrupt type. Interrupt unit **304** increments the count for the interrupt type when the signal is generated. The resulting data is stored and may be collected at a later time by the supporting software.

In one embodiment of the present invention, the mechanism of the present invention stores the count in the interrupt descriptor table (IDT). An interrupt descriptor table is a system table that associates each interrupt with a corresponding interrupt handler containing corresponding interrupt service routines. When a particular type of interrupt occurs, the interrupt unit increments the associated count. Each IDT entry is associated with a count storage area. The storage area is associated with the interrupt type, so that the interrupt unit of the present invention and associated supporting software have knowledge of the location within the IDT where the count is stored. Prior to the interrupt handler executing an interrupt service routine when a particular type of interrupt occurs, the interrupt unit of the present invention locates the memory address of the count in the IDT and increments the count value.

Turning to **Figure 4**, an exemplary diagram illustrating components for counting interrupts by type with counts stored within the IDT is depicted in accordance with a preferred embodiment of the present invention. In this example implementation, an interrupt descriptor table (IDT) **402** includes offset addresses **404** and content **406** of the IDT. In this example, the count for each interrupt type or IDT entry, VHPT data fault count **408**, is stored in IDT **402** with an offset address such as 0x0000 **410**. Corresponding code containing an interrupt service routine such as VHPT data fault code **412** is also stored in IDT **402** with the offset address of the code at 0x0008 **414**. In this example, the size of the count stored in the IDT is 8 bytes while the size of the code for each interrupt type is 400 bytes. However, the size of the code for each interrupt type may vary. Thus, the interrupt unit may locate the address of the count in the IDT for each interrupt type to increment.

In another embodiment of the present invention, the count is stored in an interrupt count table (ICT) outside of the IDT. The ICT stores count values for each interrupt type referenced by an offset address. The ICT is first allocated by supporting software and the processor is notified of the memory address of the ICT. The memory address of the ICT may be referenced through the use of a register. When a particular type of interrupt occurs, the interrupt unit goes to the memory address of the ICT and looks up the offset address of where the count is stored in the ICT for the particular interrupt type. In turn, the interrupt unit increments the count in the storage area and the processor continues to process the interrupt service routine of that type. Once the count is incremented, the supporting software may later read the value of the count and reset the count if necessary.

Turning to **Figure 5**, an exemplary diagram illustrating components for counting interrupts by type with count stored outside of IDT is depicted in accordance with a preferred embodiment of the present invention. In this example implementation, an interrupt descriptor table (IDT) **502** includes offset addresses **504**, interrupt types **506** and count offsets **508**. Interrupt types **506** represented in IDT **502** are for illustrative purpose only.

In IDT **502**, each interrupt type **506** corresponds to a offset address such as offset address **504** that identifies starting address of the associated interrupt service routine. The supporting software may be programmed with a formula such as formula **510**. Formula **510** takes the count offset value such as count offset value **508** and multiply by 8. For example, VHPT data fault interrupt has a count offset value of 0, formula **510** multiplies 0 by 8, which results a value of 0. The result is then added to memory address 0x4000 in this example to derive offset address of the count. Memory address 0x4000 is the memory address of ICT **512** indicated by a register. In this example, 0 is added to memory address 0x4000, which results an offset address 0x4000 **514**. Using formula **510** and count offset **508**, each interrupt type or IDT entry is associated with a different offset address where the count is stored. Thus, the interrupt unit may increment the count stored in the offset address calculated from formula **510** for each interrupt type.

The present invention also provides a mechanism for handling an overflow of the count. A potential overflow occurs when the number of bits used to store the count is about to wrap. The mechanism of the present invention provides logic necessary to indicate a potential overflow to the software by either setting an overflow flag in the processor status register (PSR) or by sending an interrupt signal to the processor to notify the processor an overflow may occur. The processor status register (PSR), in an Intel 64-bit processor architecture available from Intel Corporation, is a 64-bit register that maintains control information for the currently running process. As a result, if an overflow flag is set or an overflow interrupt signal is sent, the supporting software may handle the overflow accordingly, such as by reading the count and resetting the count value.

Turning next to **Figure 6**, a flowchart outlining an exemplary process of counting interrupts by type is depicted in accordance with a preferred embodiment of the present invention. The process begins when an interrupt occurs (step **602**), the interrupt unit generates an interrupt signal for a particular interrupt type (step **604**). The supporting software allocates count storage memory (step **606**) if necessary and notifies the CPU of the memory address of the count storage memory (step **608**). The interrupt unit locates the count storage offset address (step **610**) and a determination is made as whether the count is about to overflow (step **612**). If the count is about to overflow, an overflow handling routine may be invoked (step **614**) to handle the overflow to read and reset the count, the process terminating thereafter. However, if the count is not about to overflow, the interrupt unit increments the count (step **616**), the process terminating thereafter. The supporting software may later reset the count once the count is read.

Thus, the present invention provides a mechanism for counting the number of times a particular interrupt type occurs. The count may be stored in the IDT corresponding to each interrupt type or in an interrupt count table outside of the IDT. The supporting software and the interrupt unit have knowledge of memory addresses where the counts are stored. The interrupt unit may locate the count stored in a particular memory address in or outside of the IDT and increment it. In case of an overflow, an overflow handling routine may be invoke to handle the overflow accordingly.

It is important to note that while the present invention has been described in the context of a fully functioning data processing system, those of ordinary skill in the art will appreciate that the processes of the present invention are capable of being distributed in the form of a computer readable medium of instructions and a variety of forms and that the present invention applies equally regardless of the particular type of signal bearing media actually used to carry out the distribution. Examples of computer readable media include recordable-type media, such as a floppy disk, a hard disk drive, a RAM, CD-ROMs, DVD-ROMs, and transmission-type media, such as digital and analog communications links, wired or wireless communications links using transmission forms, such as, for example, radio frequency and light wave transmissions. The computer readable media may take the form of coded formats that are decoded for actual use in a particular data processing system.

The description of the present invention has been presented for purposes of illustration and description, and is not intended to be exhaustive or limited to the invention in the form disclosed. Many modifications and variations will be apparent to those of ordinary skill in the art. The embodiment was chosen and described in order to best explain the principles of the invention, the practical application, and to enable others of ordinary skill in the art to understand the invention for various embodiments with various modifications as are suited to the particular use contemplated.